Patent

Attorney Docket No.: Intel 2207/10121
**Serial No.: 09/751,762**
Assignee: Intel Corporation

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANT	:	Sailesh KOTTAPALLI et al.
SERIAL NO.	:	09/751,762
FILED	:	December 29, 2000
FOR	:	METHOD FOR CONVERTING PIPELINE STALLS CAUSED BY INSTRUCTIONS WITH LONG LATENCY MEMORY ACCESSES TO PIPELINE FLUSHES IN A MULTITHREADED PROCESSOR WHERE THE INSTRUCTIONS ARE RE-EXECUTED UPON COMPLETION OF THE ACCESSES
GROUP ART UNIT	:	2183
EXAMINER	:	Daniel H. PAN

M/S: APPEAL BRIEFS – PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

CERTIFICATION OF FACSIMILE TRANSMISSION
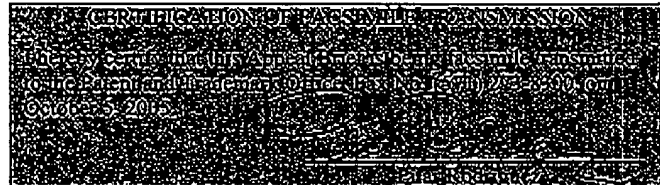
APPEAL BRIEF

Dear Sir:

This brief is in furtherance of the Notice of Appeal, filed in this case on July 5, 2005.

1.   REAL PARTY IN INTEREST

The real party in interest in this matter is Intel Corporation. (Recorded April 6, 2001, Reel/Frame 011720/0255).

2.   RELATED APPEALS AND INTERFERENCES

There are no related appeals.

3.   STATUS OF THE CLAIMS

Claims 1-21 are pending in the application. Claims 1-21 were rejected under 35 U.S.C. §103(a). This appeal is an appeal from the rejection of claims 1-21.

4.   STATUS OF AMENDMENTS

Applicants did not make any amendments to the claim subsequent to final rejection. The claims listed on page 1 of the Appendix attached to this Appeal Brief reflect the present status of the claims (including amendments entered after final rejection).

5.   SUMMARY OF THE CLAIMED SUBJECT MATTER

The embodiment of independent claim 1 generally describes a method of handling operations in a multi-threaded processing system, comprising: determining if a stalled operation of a first thread (e.g., see page 4, lines 8-10) is due to a loading of data from a memory device (e.g., Fig. 1, element 108, 109 – see page 4, lines 12); and flushing an instruction from said first thread from a pipeline of said processing system (e.g., see page 5 lines 8-16) when data is to be loaded from said memory device before executing said instruction.

The embodiment of independent claim 5 generally describes a method of handling operations in a multi-threaded processing system, comprising: determining if a stalled operation of a first thread is due to a loading of data from a memory device (e.g., see page 4 line 21- page 5 line 6); and flushing an instruction from said first thread from a pipeline of said processing system when data is to be loaded after a predetermined number of clock cycles from said memory device before said instruction can be executed (e.g., see page 5 lines 8-16).

The embodiment of independent claim 10 generally describes a processing system comprising: a scheduler (e.g., Fig. 1 element 104 – see page 5 line 17) to pass instructions from a first thread and a second thread to an execution pipeline; and pipeline control logic (e.g., Fig. 1 element 106 – see page 5 line 2) coupled to said execution pipeline to determine if a stalled execution of said first thread is due to a loading of data from a memory device (e.g., page 4 line 21- see page 5 line 6) and to flush an instruction from said first thread from said execution pipeline when data is to be loaded from said memory device before said instruction can be executed (e.g., see page 5 lines 8-16).

The embodiment of independent claim 16 generally describes a computing system comprising: a memory bus (e.g., Fig. 1, element 110, 109 – see page 4, lines 18) coupled to system memory (e.g., Fig. 1, element 108, 109 – see page 4, lines 12); and a processing system coupled to said memory bus, said processing system including a scheduler to pass instructions from a first thread and a second thread to an execution pipeline(e.g., Fig. 1 element 104 – see page 5 line 17); and pipeline control logic coupled to said execution pipeline to determine if a stalled execution of said first thread is due to a loading of data from system memory (e.g., Fig. 1 element 106 – see page 5 line 2) and to flush an instruction from said first thread from said

execution pipeline when data is to be loaded from said system memory before said instruction

can be executed (e.g., see page 5 lines 8-16).

Referring to Fig. 1, a block diagram of a processor system 100 (e.g. a microprocessor, a

digital signal processor, or the like) operated according to an embodiment of the present

invention is shown.

In this embodiment of the processor system 100, instructions are fetched by a fetch unit

101 from memory (i.e., either from L1 cache memory 108 or from L2 memory cache 109). The

scheduler controls when and in what order the instructions from thread 0 or thread 1 are supplied

to the execution ports 105. The pipeline control logic 106 monitors the execution of the threads

and tracks the return of data from L1 memory cache 108 or L2 memory cache 109 or memory

bus 110 as needed. The outputs of the execution pipes 105 are then supplied to the exception

and retirement unit 107.

According to this embodiment of the present invention, a "stall-miss-flush" approach is

used to flush and track a stalled thread from the execution pipes. In Fig. 1, in this example,

during execution in one of the execution pipes 105, a data dependent operation or instruction in

thread 0, stalls. The pipeline control logic 106, in monitoring stalled thread 0, first attempts to

retrieve the data from L1 memory cache 108, a delay of a few clock cycles. If the data is not

found in the L1 memory cache 108, the pipeline control logic 106 attempts to retrieve the data

from the L2 memory cache 109, which takes an additional number of clock cycles,

approximately 5 to 10 cycles in this embodiment of the processor system 100. At the same time,

stalled thread 0 blocks the pipeline for non-stalling thread 1 that follows in the execution pipes

105. The pipeline control logic 106 flags the instruction of the thread as a missed search (i.e., a

"miss"). The stall on thread 0 is then released and the instruction is flushed through the

execution pipes 105 into the exception and retirement logic 107, which allows thread 1 to

continue its execution down the pipeline. Pipeline control logic 106 continues to retrieve the

data from a third level of memory (e.g., random access memory coupled to memory bus 110),

which may take several hundreds of clock cycles to recover. Exception and retirement logic 107

detects that thread 0 has data dependencies that have not been completed and informs fetch unit

101 to fetch the miss instruction from thread 0 once more and attempt to execute the instructions

from thread 0 again.

This time around, when the miss instruction from thread 0 reaches scheduler 104, the

scheduler 104 awaits an update pipeline control logic 106 when the data needed by thread 0 has

been retrieved from the memory bus. When the data is available, the scheduler starts dispersal of

this instruction from thread 0 into the execution pipes 105. This time through, thread 0 can

resolve its data dependency without stalling by accessing the data made available.

An example of the operation of pipeline control logic 107 in this embodiment is shown in

Fig. 2.

6.   GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A.   Are claims 1-21 rendered obvious under 35 U.S.C. §103(a) by Parady (U.S.

Patent No. 5,933,627) in view of Hennessy (Computer Organization and Design)?

7.   ARGUMENT

A.   Claims 1-21 are not rendered obvious under §103(a) by Parady in view of
      Hennessey

Applicants respectfully submit that none of the cited sections of Parady teach, suggest or

reflect at least "[a] method of handling operations in a multi-threaded processing system,

Serial No. 09/751,762
Appeal Brief Under 37 CFR 41.37 Filed September 5, 2005
Advisory Action dated May 23, 2005

comprising determining if a stalled operation of a first thread is due to a loading of data from a

memory device *and flushing an instruction from said first thread from a pipeline of said*

processing system when data is to be loaded from said memory device before executing said

instruction" [e.g., as described in the embodiment of independent claim 1].

Applicants submit their concurrence with the Examiner's assertion Parady *does not*

disclose flushing an instruction from said first thread from a pipeline of said processing system

when data is to be loaded from said memory device before executing such instruction (see page

2, 5 and 8 at least of the previous Office Action).

Despite this assertion, the Examiner in the Final Office Action (see page 14) states that

Parady has shown in column 4, lines 45-51 an embodiment where control is immediately

transferred back to the thread that executed a load when the data is ready before other threads

can advance. This means that an instruction about to be dispatched would have been flushed on

a switch to a critical thread because data is to be loaded for execution of an instruction in the

critical thread before the flushed instruction can execute. Applicants respectfully disagree.

Column 4, lines 45-51 state

> Alternately, a particular thread could be identified as a critical thread, and generate an
> interrupt as soon as the memory access is completed. The returned data from the load must be
> provided to the appropriate register for the thread which requested it. This could be done by
> using separate load buffers for each thread, or by storing a two bit tag in the load buffer
> indicating the appropriate thread.

First, as is clear from the cited section itself, there is no literal basis for the Examiner's

assertions. The Parady reference merely discloses a critical thread's ability to generate an

interrupt after a memory access has been completed. The returned data is then sent to the

appropriate register through the use of separate load buffers or by storing a two bit tag in the load

buffer. However, nowhere in this section is there the teaching, suggestion or disclosure of

-6-

flushing an instruction from said first thread from a pipeline of said processing system when data is to be loaded from said memory device before executing said instruction, affirming the Office Action's stated position detailed above.

Second, regardless of the fact that the Examiner's initial rejection did not include the assertion, the Examiner asserts by implication that an instruction about to be dispatched *would have been* flushed on a switch to a critical thread because data is to be loaded for execution of an instruction in the critical thread before the flushed instruction can execute. Applicants again respectfully disagree. Above and beyond the fact that Examiner's assertion are totally unsupported by the text of Parady, Applicants submit a particular thread's ability to generate a interrupt does not at all implicate the flushing of an instruction from a first thread from a pipeline when data is to be loaded from said memory device before executing said instruction as specifically recited by independent claim 1. The Parady interrupt may implicate many possibilities (e.g., a delay), which Applicants are unwilling and not compelled to speculate upon. The Parady reference clearly does not specifically disclose the limitations of independent claim 1, and the Office Action unsupported hypothetical assertions are inadequate to support a proper 35 U.S.C. 103(a) rejection.

Furthermore, in the Advisory Action, the Examiner now asserts two new sections of Parady not previously asserted. He asserts Parady does disclose the flushing from the memory (e.g., the register files) before execution at col. 5 lines 30-37, and cites to col. 3, lines 12-14 for background. Applicants again disagree. Column 3 lines 12-14 state:

> Dispatch unit 28 will provide four decoded instructions at a time along a bus 30, each instruction being provided to one of eight functional units 32-46.

-7-

Serial No. 09/751,762
Appeal Brief Under 37 CFR 41.37 Filed September 5, 2005
Advisory Action dated May 23, 2005

Applicants submit this section clearly does not offer any background pertaining to

flushing of instructions, but merely details the operation of a dispatch unit with respect to

functional units 32-46. Column 5, lines 30-37 state:

> Register files often have 5 or more ports, and the ports can take up more silicon than
> simply duplicating the registers. Thus, it may be more economical to swap the thread data
> in and out of the four, single or dual-ported shadow register files. This can require two
> ports to be added to the main register file, if the loading and flushing of register files is
> not dispatched through existing functional units such as the load/store unit.

This section is intended to disclose the operation of register files, and specifically

swapping of thread data in and out of register. At best, this section also discloses the concept of

"flushing" generically. However, it is clear that this section does not disclose flushing an

instruction from said first thread from a pipeline of said processing system when data is to be

loaded from said memory device before executing said instruction as specifically recited in the

embodiment of claim 1. Again, the Parady reference falls short of supporting a proper 35 U.S.C.

103(a) reference.

Lastly, in its original 35 U.S.C. §103(a) rejection (prior to the more recent arguments by

implication using Parady), the Examiner cites page 505 of Hennessey as showing flushing an

instruction from a pipeline of a processing system. Applicants respectfully disagree and submit

Hennessey only discloses the flushing of instructions generally. In fact, the only line of

Hennessey disclosing "flushing" at all is first line of paragraph 3 on page 505, which states:

"[w]e already saw how to flush the instruction in the IF stage by turning it into a nop". Clearly,

this is inadequate to disclose the flushing of an instruction when data is to be loaded from the

memory device in order to execute an instruction as described in the embodiment of claim 1.

Therefore, Applicants submit since each and every limitations is not found in the cited

references, the cited references Parady and Hennessey cannot be combined to adequately form

-8-

Serial No. 09/751,762
Appeal Brief Under 37 CFR 41.37 Filed September 5, 2005
Advisory Action dated May 23, 2005

the basis of a proper 35 U.S.C. §103(a) rejection of independent claim 1. Independent claims 5,

10, and 16 contain substantively similar limitations and therefore are also allowable for similar

reasons. Claims 2-4, 6-9, 11-15 and 17-21 depend from allowable independent claims 1, 5, 10

and 16, and therefore are in condition for allowance as well.

Appellants therefore respectfully request that the Board of Patent Appeals and

Interferences reverse the Examiner's decision rejecting claims 1-21 and direct the Examiner to
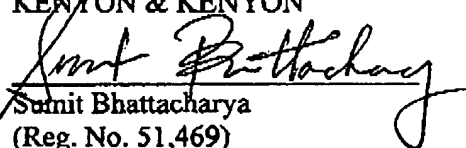
pass the case to issue.

The Examiner is hereby authorized to charge the appeal brief fee of $500.00 and any

additional fees which may be necessary for consideration of this paper to Kenyon & Kenyon

Deposit Account No. **11-0600**.

Respectfully submitted,

KENYON & KENYON

Date: October 5, 2005   By: Sumit Bhattacharya
                                    (Reg. No. 51,469)

KENYON & KENYON
333 West San Carlos St., Suite 600
San Jose, CA 95110
Telephone: (408) 975-7500
Facsimile: (408) 975-7501

75864

-9-

APPENDIX

(Brief of Appellants Sailesh Kottapalli et al.
U.S. Patent Application Serial No. 09/751,762)

8. CLAIMS ON APPEAL

1.   (Previously Presented) A method of handling operations in a multi-threaded

processing system, comprising:

determining if a stalled operation of a first thread is due to a loading of data from a

memory device; and

flushing an instruction from said first thread from a pipeline of said processing system

when data is to be loaded from said memory device before executing said instruction.

2.   (Original) The method of claim 1 wherein said memory device is system memory

coupled to a memory bus.

3.   (Original) The method of claim 1 further comprising:

marking said instruction as a miss.

4.   (Original) The method of claim 3 further comprising:

rescheduling said instruction to be executed in said pipeline.

5.   (Original) A method of handling operation in a multi-threaded processing system,

comprising:

-1-

determining if a stalled operation of a first thread is due to a loading of data from a

memory device; and

flushing an instruction from said first thread from a pipeline of said processing system

when data is to be loaded after a predetermined number of clock cycles from said memory device

before said instruction can be executed.

6.   (Original) The method of claim 5 wherein said memory device is system memory

coupled to a memory bus.

7.   (Original) The method of claim 6 further comprising:

marking said instruction as a miss.

8.   (Original) The method of claim 7 further comprising:

rescheduling said instruction to be executed in said pipeline.

9.   (Original) The method of claim 8 further comprising:

executing said instruction when data is loaded from said memory device.

10.   (Previously Presented) A processing system comprising:

a scheduler to pass instructions from a first thread and a second thread to an execution

pipeline; and

pipeline control logic coupled to said execution pipeline to determine if a stalled

execution of said first thread is due to a loading of data from a memory device and to flush an

-2-

instruction from said first thread from said execution pipeline when data is to be loaded from said memory device before said instruction can be executed.

11.   (Original) The processing system of claim 10 wherein said pipeline control logic is to mark said instruction as a miss.

12.   (Original) The processing system of claim 10 further comprising:

an exception and retirement logic coupled to said execution pipeline.

13.   (Original) The processing system of claim 12 wherein said instruction marked as a miss is to be detected by said exception and retirement logic.

14.   (Original) The processing system of claim 13 further comprising:

a fetch unit to provide said instruction to said scheduler.

15.   (Original) The processing system of claim 14 wherein said pipeline control logic is to cause said instruction to be executed when data is loaded from said memory device.

16.   (Previously Presented) A computing system comprising:

a memory bus coupled to system memory; and

a processing system coupled to said memory bus, said processing system including

a scheduler to pass instructions from a first thread and a second thread to an

execution pipeline; and

-3-

pipeline control logic coupled to said execution pipeline to determine if a stalled

execution of said first thread is due to a loading of data from system memory and to flush

an instruction from said first thread from said execution pipeline when data is to be

loaded from said system memory before said instruction can be executed.

17.   (Original) The computing system of claim 16 wherein said pipeline control logic is to

mark said instruction as a miss.

18.   (Previously Presented) The computing system of claim 16 wherein said processing

system further includes

        an exception and retirement logic coupled to said execution pipeline.

19.   (Original) The computing system of claim 18 wherein said instruction marked as a miss

is to be detected by said exception and retirement logic.

20.   (Original) The computing system of claim 19 wherein said processing system further

includes

        a fetch unit to provide said instruction to said scheduler.

21.   (Original) The computing system of claim 20 wherein said pipeline control logic is to

cause said instruction to be executed when data is loaded from said system memory.

-4-

9. EVIDENCE APPENDIX

No further evidence has been submitted with this Appeal Brief.

-5-

10. RELATED PROCEEDINGS APPENDIX

Per Section 2 above, there are no related proceedings to the present Appeal.

-6-